

REMARKS

Claims 1-32 are currently pending in the subject application. Applicants thank the Examiner for the telephonic interview conducted on Tuesday, July 12, 2005 in which the 112 rejection and 103 rejections were discussed. The Examiner suggested that claims 1 and 17 be clarified.

Claims 1, 5, 9, 17, 21, and 25 have been amended.

I 112 Rejection

Claims 1 and 17 stand rejected under 35 U.S.C. 112, second paragraph as being indefinite for failing to particularly point out and distinctly claim the subject matter regarded as the invention because “‘the method performing no more than one scan per table’ cannot be done in this invention. ...the destination table requires two scans in order to produce the join result set and using the join result to update/insert the destination table” as stated on page 2, section 3 in the Office Action mailed 04/21/05.

Applicants respectfully traverse. Applicants point to specification page 7 line 21 through page 14, line 18, and to Figure 4 for support for “the method performing no more than one scan per table. Claims 1 and 17 have been amended to recite “the outer join designating the source table as being preserved” to help clarify the claims as suggested by the Examiner in the telephonic interview. The outer join result table is made up of the comparison of the source and destination including all rows in the source that match the destination and all the rows in the source that do not have a match in the destination table (the source table is preserved). Consequently, the outerjoin result includes rows to be updated (matching) in the destination table and rows to be added into the destination table (source rows preserved). This outerjoin result along with the other limitations in the claims of the instant application require only one scan for each table.

II. 103 Rejections

A. Claims 1-32 stand rejected under 35 U.S.C. § 103 as being unpatentable over U.S. Patent 6,581,205 issued June 17 2003, to Cochrane et al. (hereafter Cochrane) in view of U.S. Patent 6,735,587 issued May 11, 2004 to Colby et al. (hereafter Colby).

1. Amended claim 1 recites "[a] method for applying a row from a source table to a destination table, the method comprising: selecting a first column from a source table; selecting a second column from a destination table; performing an outer join operation on the source table and the destination table using the first and second columns, the outer join designating the source table as being preserved; updating each row in the destination table with a row from the results of the outer join operation containing a matching element in the first and second columns; and inserting into the destination table each row from the results of the outer join operation with a non-matching element in the first and second columns, the method performing no more than one scan per table." In contrast, Cochrane teaches a method for maintaining a materialized view (MV) derived from at least one base table in a database. When an update is performed to the base table in a transaction, the materialized view must be updated as well. During the updating process multiple scans of the base or destination table occur. For example, Cochrane discloses:

Consider the following materialized view (MV):

MV:
SELECT A, SUM(SALES) AS S, COUNT(*) AS C
FROM T
GROUP BY A

Assume that a number of rows are inserted into T, wherein these rows are denoted as DELTA-T. There are two possible effects on MV, depending on the value of column A after the grouping on column A:

The first operation, the INSERT, returns any non-matching groups, whereas the second operation, the UPDATE, returns all matching groups. These two operations can be combined together into an outerjoin, which returns both matching and non-matching pairs.

Consider the output of the following outerjoin:

```
OJ:
SELECT Q.A, Q.S, Q.C, MV.C AS INDICATOR
FROM TABLE (SELECT A, SUM(SALES) AS S,
COUNT(*) AS C
FROM DELTA-T
GROUP BY A) AS Q(A,S,C)
LEFT JOIN MV ON Q.A=MV.A
```

That is, the DELTA-T is first grouped on column A, and then an outerjoin is performed with MV. The derived table Q(A,S,) is the tuple-preserving operand, whereas MV is the null-producing operand. With the outerjoin operation, both matching as well as non-matching rows with respect to the grouping column A of the DELTA-T are returned. By definition of the outerjoin, the output column MV.C (null-producing operand) is null when there is a row in Q(A,S,C) that does not have a matching value of A in MV.

Hence, the value in MV.C can be used as an indicator to separate out the matching and non-matching rows.

MATCHING ROWS:

```
SELECT A, S, C
FROM OJ
WHERE INDICATOR IS NOT NULL
```

NON-MATCHING ROWS:

```
SELECT A, S, C
FROM OJ
WHERE INDICATOR IS NULL
```

As a result, both the matching and non-matching pairs can be computed in a single operation using an outerjoin.

(Cochrane, column 6 line 15 through column 7, line 20) underline added.

Cochrane further discloses:

When the underlying base tables 404 are modified, the summary table 400 must also be maintained to accurately reflect the modified tables. The results of the specification query 402 are joined 406 to the summary table 400 to determine whether an UPDATE 408, DELETE 410, or INSERT 412 operation is required to maintain the summary table 400.

(Cochrane, column 5, lines 15-19) underline added.

These passages of Cochrane show the use of a single operation (outerjoin statement) to identify and return both the matching and non matching rows. The result of the outerjoin determine whether an update, delete, or insert is required. However, Cochrane is silent as to performing the actual updating, deleting, or inserting. Since Cochrane utilizes one statement to identify the matching and non-matching rows, any further actions, such as the updating or inserting, require additional statements. One skilled in the art understands, as revealed on page 3 of the Background of the disclosure, that these additional statements cause additional scans of each subject table. One scan is required for the outerjoin, and an additional scan is required for *each* of the UPDATE, DELETE or INSERT operations. As Cochrane discloses *multiple query language statements performing multiple scans per table* to enact the process, Cochrane does not disclose "[a] method for applying a row from a source table to a destination table, the method comprising: selecting a first column from a source table; selecting a second column from a destination table; performing an outer join operation on the source table and the destination table using the first and second columns, the outer join designating the source table as being preserved; updating each row in the destination table with a row from the results of the outer join operation containing a matching element in the first and second columns; and inserting into the destination table each row from the results of the outer join operation with a non-matching element in the first and second columns, the method performing no more than one scan per table" as recited in amended claim 1. Colby does not cure these deficiencies.

Colby teaches a method for maintaining pre-computed aggregated views. Colby specifically discloses:

A pre-computed aggregate view typically has an associated pre-computed aggregate table, which contains the materialized (computed) results of the view. The term V refers to a pre-computed view and V.sub.D and MV refer to a pre-computed view definition (i.e., query) and a result table in which the view is materialized (i.e., a pre-computed aggregate table), respectively. Maintenance or modification of a pre-computed view refers to maintenance or modification, respectively, of the pre-computed view's associated pre-computed table. It is assumed that a pre-computed aggregate table has a column that contains unique values. In one implementation, the column includes physical row identifiers. In another implementation, the column

includes a primary key. It is assumed that such a column is called "rowid." The tables in the database can contain duplicates, as can the change sets.
(Colby, column 4, lines 49-65) underline added.

Colby further discloses:

FIG. 2 shows another method in accordance with the invention for incrementally maintaining a pre-computed aggregate view that is self-maintainable. The system receives the pre-computed aggregate view V that is self-maintainable, including its view definition V.sub.D and materialized aggregate table MV (step 202). The system also receives changes to a base table of the pre-computed aggregate view, the changes being represented as deletions and insertions (step 204). The system tags insertions and deletions with distinguishing literals and combines them to produce .delta.F (deltaF) (step 206). The system computes aggregations on .delta.F to produce aggregated change set .delta.G (deltaG) (step 208). The system matches rows from .delta.G with rows in MV and produces .delta.J (deltaJ), which contains all rows from .delta.G with matched rows tagged with information from corresponding rows in MV and further contains non-matching rows from .delta.G tagged as non-matching (step 210). (Matched rows are rows that have corresponding rows and non-matching rows are rows that do not have corresponding rows.) One way to tag rows in .delta.J as non-matching is to insert NULL values in MV columns. The system produces .delta.J' (deltaJ') by selecting from .delta.J rows identified as either: (i) matched rows or (ii) identified as non-matching but resulting from more base table changes that are insertions than deletions for the aggregated group represented by the row (step 212). The system inserts new rows into MV, the new rows being constructed from rows (in .delta.J') identified as non-matching rows (step 214). The system removes the identified rows from .delta.J' to produce .delta.I (deltaI) (also step 214). The system deletes from MV rows matching rows in .delta.I, each row representing an aggregated group that has as many base table deletions as the sum of the number of base table insertions and the number of pre-modified base table rows for that group as stored in MV (step 216). The system removes these identified rows from .delta.I to produce .delta.D (deltaD) (also step 216). The system updates rows in MV matching rows in .delta.D (step 218).

The following describes in detail the method shown in FIG. 2 for incrementally maintaining a pre-computed aggregate view in the presence of non-minimal changes.

1. .delta.G is computed as follows: Let V'.sub.D be obtained by modifying V.sub.D by replacing F in V.sub.D with .delta.F and replacing each aggregation expression a.sub.i with three functions a.sub.i.sup.+, a.sub.i.sup.-, and a.sub.i.sup.d defined as follows:
(a) if a.sub.i is count, then a.sub.i.sup.+ is count(case when f=x then z else NULL end), and a.sub.i.sup.- is count(case when f=y then z else NULL end) where z is expr if the aggregation expression is count(expr) and is some non-null literal, otherwise

(i.e., if it is count(*)).

(b) if a.sub.i is sum(expr), then a.sub.i.sup.+ is sum(case when f=x then expr else NULL end), and a.sub.i.sup.- is sum(case when f=y then expr else NULL end).

(c) ad is defined as follows:

a.sub.i.sup.d=a.sub.i.sup.+ -a.sub.i.sup.-, if a.sub.i is count

=case when isnull(a.sub.i.sup.-) then a.sub.i.sup.+

when isnull(a.sub.i.sup.-) then a.sub.i.sup.+

else a.sub.i.sup.+ -a.sub.i.sup.- end, if a.sub.i is sum.

Note that a .sub.i.sup.+ represents contributions from changes that are insertions and a.sub.i.sup.- represents contribution from changes that are deletions. Note further that the use of conditional case expressions inside aggregation functions (e.g., such as the use described in (a)-(b) above) requires the addition of only one column to the original set of insertions and deletions in the construction of .delta.F. The technique described in the previous two notes also apply to the method (described in the next section) for maintaining either pre-computed aggregate views that are self-maintainable or pre-computed aggregate views that are not self-maintainable. .delta.G is computed as V'.sub.D (.delta.F).

2. .delta.J is computed as .delta.G{character pullout}.sub.p MV where the outerjoin condition, p, is:

.delta.F.g.sub.1 {character pullout}MV.g.sub.1 {character pullout}, ..., {character pullout}.delta.F.g.sub.m {character pullout}MV.g.sub.m.

The {character pullout} operator may be replaced by a regular equality=operator for each grouping column g.sub.i that is not nullable (guaranteed to not have null values).

3. Let c denote one of the count(*) or count() expressions and let c.sup.+ and c.sup.- denote the corresponding a.sub.i.sup.+ and a.sub.i.sup.- expressions described in 1(a) and 1(b) above. .delta.J is computed as .sigma..sub.p (.delta.J) where p is (isnotnull(MV.rowid)){character pullout}(c.sup.+ >c.sup.-) and MV.rowid is the rowid (which can actually be any non-nullable) column of MV (in the input .delta.J).

4. .delta.I is computed as I.sub.p.m (MV, .delta.J) where p is isnull(MV.rowid) and the mapping m of expressions from source columns of J.delta.' to columns of the target of the insert, MV is as follows: Each grouping column from the .delta.G component of .delta.J maps to the corresponding grouping column in MV. For each

aggregation column $a.sub.i$ in $V.sub.D$, the corresponding $a.sub.i.sup.d$ column (from the $\delta.G$ component) of $\delta.J'$ maps to the corresponding aggregation column in MV. Note that the reference to MV.rowid in the predicate p is to the MV.rowid column in the input $\delta.J'$.

$\delta.D$ is computed as $D.sub.MV.rowid.p (MV, \delta.I)$ where p is $c.sup.- = MV.c + c.sup.+$.

$\delta.U$ is computed as $U.sub.MV.rowid.TRUE.m (MV, \delta.D)$ where the mapping m of source expressions to target columns is as follows:

Each grouping column $g.sub.i$ in $\delta.D$ maps to the corresponding grouping column in MV. For each aggregation column $a.sub.i$ in $V.sub.D$, the corresponding $u.sub.a.sub..sub.i$, as defined below, maps to the corresponding aggregation column $a.sub.i$ in MV.

$u.sub.a.sub..sub.i$ is as follows:

$u.sub.a.sub..sub.i = MV.a.sub.i + a.sub.i.sup.+ - a.sub.i.sup.-$, if $a.sub.i$ is $count(expr)$ or $count(*)$

$= null-sub(null-add(MV.a.sub.i, a.sub.i.sup.+), a.sub.i.sup.-)$ if $a.sub.i$ is $sum(expr)$

The previous delete can be modified so that the deleted rows are not in the delete's result set. However, because they are deleted anyway, they are irrelevant for the next update operation.

(Colby, column 7, line 62 through column 9, line 31) underline added.

These passages of Colby teach a method of maintaining pre-computed aggregated view (MV) to be consistent with changes to a base table from which the MV was constructed. The maintenance process uses a six step procedure including the production of a plurality of delta tables ($\delta.D$, $\delta.F$, $\delta.G$, $\delta.J$, $\delta.J'$, $\delta.I$, $\delta.U$). The computation of each of the delta tables is accomplished via query language statements, as shown by Colby. Many of these statements compare MV to the various delta tables, causing additional scans of the MV table. Thus, Colby scans the MV, or the destination table, more than once. As such, Colby does not teach "performing no more than one scan per table" as recited in amended claim 1.

Cochrane and Colby, neither alone nor together, disclose, teach or suggest all the limitations in claim 1 and therefore, cannot be used to preclude patentability of claim 1 under U.S.C. § 103.

2. Claim 17 recites sufficiently similar limitations to claim 1, and as such, is patentable over Cochrane and Colby for at least the same reasons as claim 1.
3. Claims 2-4 and 18-20 depend on claim 1 and 17, and as such, are patentable over Cochrane and Colby for at least the same reasons as those claims.
4. Amended claim 5 is directed to a statement, but it possess the common limitation to claims 1 and 17 of “performing no more than one scan per table.” Section A 1 of this paper shows that Cochrane and Colby do not disclose, teach, or suggest, performing the method with “no more than one scan per table.” As such Cochrane and Colby do not teach all the limitations in claim 5, and cannot be used to preclude patentability of claim 5 under 35 U.S.C. § 103.
5. Claim 9 recites “[a] method for upserting a source table with a destination table, the method comprising: selecting from a source table a first column comprising a plurality of elements; selecting from a destination table a second column comprising a plurality of elements; updating a row in the destination table with a row from the source table upon the success of a comparison operation on an element in the first column of the row from the source table and an element in the second column of the row from the destination table; and inserting a row from the source table into the destination table upon the failure of a comparison operation on an element in the first column of the row from the source table and an element in the second column of the row from the destination table, the method using no more than one query language statement.” In contrast, Cochrane teaches a method for maintaining a materialized view (MV) derived from at least one base table in a database. When an update is performed to the base table in a transaction, the materialized view must be updated as well. During the updating process multiple scans of the base or destination table occur. For example, Cochrane discloses:

Consider the following materialized view (MV):

MV:
SELECT A, SUM(SALES) AS S, COUNT(*) AS C
FROM T

GROUP BY A

Assume that a number of rows are inserted into T, wherein these rows are denoted as DELTA-T. There are two possible effects on MV, depending on the value of column A after the grouping on column A:

The first operation, the INSERT, returns any non-matching groups, whereas the second operation, the UPDATE, returns all matching groups. These two operations can be combined together into an outerjoin, which returns both matching and non-matching pairs.

Consider the output of the following outerjoin:

```
OJ:
SELECT Q.A, Q.S, Q.C, MV.C AS INDICATOR
FROM TABLE (SELECT A, SUM(SALES) AS S,
COUNT(*) AS C
FROM DELTA-T
GROUP BY A) AS Q(A,S,C)
LEFT JOIN MV ON Q.A=MV.A
```

That is, the DELTA-T is first grouped on column A, and then an outerjoin is performed with MV. The derived table Q(A,S,) is the tuple-preserving operand, whereas MV is the null-producing operand. With the outerjoin operation, both matching as well as non-matching rows with respect to the grouping column A of the DELTA-T are returned. By definition of the outerjoin, the output column MV.C (null-producing operand) is null when there is a row in Q(A,S,C) that does not have a matching value of A in MV.

Hence, the value in MV.C can be used as an indicator to separate out the matching and non-matching rows.

MATCHING ROWS:

```
SELECT A, S, C
FROM OJ
WHERE INDICATOR IS NOT NULL
```

NON-MATCHING ROWS:

```
SELECT A, S, C
FROM OJ
WHERE INDICATOR IS NULL
```

As a result, both the matching and non-matching pairs can be computed in a single operation using an outerjoin.
(Cochrane, column 6 line 15 through column 7, line 20) underline added.

Cochrane further discloses:

When the underlying base tables **404** are modified, the summary table **400** must also be maintained to accurately reflect the modified tables. The results of the specification query **402** are joined **406** to the summary table **400** to determine whether an UPDATE **408**, DELETE **410**, or INSERT **412** operation is required to maintain the summary table **400**.
(Cochrane, column 5, lines 15-19) underline added.

These passages of Cochrane show the use of a single operation (outerjoin statement) to identify and return both the matching and non matching rows. The matching rows determine whether an update, delete, or insert is required. However, Cochrane is silent as to performing the actual updating, deleting, or inserting. Since Cochrane utilizes one statement to identify the matching and non-matching rows, any further actions, such as the updating or inserting, require additional statements. As Cochrane discloses *multiple query language statements*, Cochrane does not disclose "[a] method for upserting a source table with a destination table, the method comprising: selecting from a source table a first column comprising a plurality of elements; selecting from a destination table a second column comprising a plurality of elements; updating a row in the destination table with a row from the source table upon the success of a comparison operation on an element in the first column of the row from the source table and an element in the second column of the row from the destination table; and inserting a row from the source table into the destination table upon the failure of a comparison operation on an element in the first column of the row from the source table and an element in the second column of the row from the destination table, the method using no more than one query language statement" as recited in amended claim 9. Colby does not cure these deficiencies.

Colby teaches a method for maintaining pre-computed aggregated views. Colby specifically discloses:

A pre-computed aggregate view typically has an associated pre-computed aggregate table, which contains the materialized (computed) results of the view. The term V refers to a pre-computed view and V.sub.D and MV refer to a pre-computed view definition (i.e., query) and a result table in which the view is materialized (i.e., a pre-computed aggregate table), respectively. Maintenance or modification of a pre-computed view refers to maintenance or modification, respectively, of the pre-computed view's associated pre-computed table. It is assumed that a pre-computed aggregate table has a column that contains unique values. In one implementation, the column includes physical row identifiers. In another implementation, the column includes a primary key. It is assumed that such a column is called "rowid." The tables in the database can contain duplicates, as can the change sets.

(Colby, column 4, lines 49-65) underline added.

Colby further discloses:

FIG. 2 shows another method in accordance with the invention for incrementally maintaining a pre-computed aggregate view that is self-maintainable. The system receives the pre-computed aggregate view V that is self-maintainable, including its view definition V.sub.D and materialized aggregate table MV (step 202). The system also receives changes to a base table of the pre-computed aggregate view, the changes being represented as deletions and insertions (step 204). The system tags insertions and deletions with distinguishing literals and combines them to produce .delta.F (deltaF) (step 206). The system computes aggregations on .delta.F to produce aggregated change set .delta.G (deltaG) (step 208). The system matches rows from .delta.G with rows in MV and produces .delta.J (deltaJ), which contains all rows from .delta.G with matched rows tagged with information from corresponding rows in MV and further contains non-matching rows from .delta.G tagged as non-matching (step 210). (Matched rows are rows that have corresponding rows and non-matching rows are rows that do not have corresponding rows.) One way to tag rows in .delta.J as non-matching is to insert NULL values in MV columns. The system produces .delta.I' (deltaJ') by selecting from .delta.J rows identified as either: (i) matched rows or (ii) identified as non-matching but resulting from more base table changes that are insertions than deletions for the aggregated group represented by the row (step 212). The system inserts new rows into MV, the new rows being constructed from rows (in .delta.J') identified as non-matching rows (step 214). The system removes the identified rows from .delta.J' to produce .delta.I (deltaI) (also step 214). The system deletes from MV rows matching rows in .delta.I, each row representing an aggregated group that has as many base table deletions as the sum of the number of base table insertions and the number of pre-modified base table rows for that group as stored in MV (step 216). The system removes these identified rows from .delta.I to produce .delta.D (deltaD) (also step 216). The system updates rows in MV matching rows in .delta.D (step 218).

The following describes in detail the method shown in FIG. 2 for incrementally

maintaining a pre-computed aggregate view in the presence of non-minimal changes.

1. ΔG is computed as follows: Let $V'.sub.D$ be obtained by modifying $V.sub.D$ by replacing F in $V.sub.D$ with ΔF and replacing each aggregation expression $a.sub.i$ with three functions $a.sub.i.sup.+$, $a.sub.i.sup.-$, and $a.sub.i.sup.d$ defined as follows:

(a) if $a.sub.i$ is count, then $a.sub.i.sup.+$ is count(case when $f=x$ then z else NULL end), and $a.sub.i.sup.-$ is count(case when $f=y$ then z else NULL end) where z is expr if the aggregation expression is count(expr) and is some non-null literal, otherwise (i.e., if it is count(*)).

(b) if $a.sub.i$ is sum(expr), then $a.sub.i.sup.+$ is sum(case when $f=x$ then expr else NULL end), and $a.sub.i.sup.-$ is sum(case when $f=y$ then expr else NULL end).

(c) $a.d$ is defined as follows:

$a.sub.i.sup.d = a.sub.i.sup.+ - a.sub.i.sup.-$, if $a.sub.i$ is count

$= \text{case when isnull}(a.sub.i.sup.-) \text{ then } a.sub.i.sup.+$

$\text{when isnull}(a.sub.i.sup.-) \text{ then } a.sub.i.sup.+$

$\text{else } a.sub.i.sup.+ - a.sub.i.sup.- \text{ end, if } a.sub.i \text{ is sum.}$

Note that $a.sub.i.sup.+$ represents contributions from changes that are insertions and $a.sub.i.sup.-$ represents contribution from changes that are deletions. Note further that the use of conditional case expressions inside aggregation functions (e.g., such as the use described in (a)-(b) above) requires the addition of only one column to the original set of insertions and deletions in the construction of ΔF . The technique described in the previous two notes also apply to the method (described in the next section) for maintaining either pre-computed aggregate views that are self-maintainable or pre-computed aggregate views that are not self-maintainable. ΔG is computed as $V'.sub.D(\Delta F)$.

2. ΔJ is computed as $\Delta G\{\text{character pullout}\}.sub.p$ MV where the outerjoin condition, p , is:

$\Delta F.g.sub.1\{\text{character pullout}\}.MV.g.sub.1\{\text{character pullout}\}, \dots, \{\text{character pullout}\}.\Delta F.g.sub.m\{\text{character pullout}\}.MV.g.sub.m$

The $\{\text{character pullout}\}$ operator may be replaced by a regular equality=operator for each grouping column $g.sub.i$ that is not nullable (guaranteed to not have null values).

3. Let c denote one of the $\text{count}(\ast)$ or $\text{count}()$ expressions and let $c.\text{sup.}+$ and $c.\text{sup.}-$ denote the corresponding $a.\text{sub.i}.\text{sup.}+$ and $a.\text{sub.i}.\text{sup.}-$ expressions described in 1(a) and 1(b) above. delta.J is computed as $\text{sigma}.\text{sub.p}(\text{delta.J})$ where p is $(\text{isnotnull}(\text{MV}.\text{rowid}))\{\text{character pullout}\}(c.\text{sup.}+ > c.\text{sup.}-)$ and $\text{MV}.\text{rowid}$ is the rowid (which can actually be any non-nullable) column of MV (in the input delta.J).

4. delta.I is computed as $\text{I}.\text{sub.p.m}(\text{MV}, \text{delta.J})$ where p is $\text{isnull}(\text{MV}.\text{rowid})$ and the mapping m of expressions from source columns of $\text{J}.\text{delta.}$ to columns of the target of the insert, MV is as follows: Each grouping column from the delta.G component of delta.J maps to the corresponding grouping column in MV. For each aggregation column $a.\text{sub.i}$ in $\text{V}.\text{sub.D}$, the corresponding $a.\text{sub.i}.\text{sup.d}$ column (from the delta.G component) of delta.J maps to the corresponding aggregation column in MV. Note that the reference to $\text{MV}.\text{rowid}$ in the predicate p is to the $\text{MV}.\text{rowid}$ column in the input delta.J .

5. delta.D is computed as $\text{D}.\text{sub.MV}.\text{rowid.p}(\text{MV}, \text{delta.I})$ where p is $c.\text{sup.}- = \text{MV}.c + c.\text{sup.}+$.

6. delta.U is computed as $\text{U}.\text{sub.MV}.\text{rowid.TRUE.m}(\text{MV}, \text{delta.D})$ where the mapping m of source expressions to target columns is as follows:

Each grouping column $g.\text{sub.i}$ in delta.D maps to the corresponding grouping column in MV. For each aggregation column $a.\text{sub.i}$ in $\text{V}.\text{sub.D}$, the corresponding $u.\text{sub.a}.\text{sub.sub.i}$, as defined below, maps to the corresponding aggregation column $a.\text{sub.i}$ in MV.

$u.\text{sub.a}.\text{sub.sub.i}$ is as follows:

$u.\text{sub.a}.\text{sub.sub.i} = \text{MV}.a.\text{sub.i} + a.\text{sub.i}.\text{sup.}+ - a.\text{sub.i}.\text{sup.}-$, if $a.\text{sub.i}$ is $\text{count}(\text{expr})$ or $\text{count}(\ast)$

$= \text{null} - \text{sub}(\text{null} - \text{add}(\text{MV}.a.\text{sub.i}, a.\text{sub.i}.\text{sup.}+), a.\text{sub.i}.\text{sup.}-)$ if $a.\text{sub.i}$ is $\text{sum}(\text{expr})$

The previous delete can be modified so that the deleted rows are not in the delete's result set. However, because they are deleted anyway, they are irrelevant for the next update operation.

(Colby, column 7, line 62 through column 9, line 31) underline added.

These passages of Colby teach a method of maintaining pre-computed aggregated view (MV) to be consistent with changes to a base table from which the MV was constructed. The maintenance process uses a *six* step procedure including the production of a plurality of delta tables (deltaD ,

deltaF, deltaG, deltaJ, deltaJ', deltaI, deltaU). The computation of each of the delta tables is accomplished via separate query language statements, as shown by the passages of Colby cited above. Colby teaches the use of conditional case expressions but limits their use to aggregation functions. As only step 1 performs an aggregation function, Colby limits the conditional case expression to step 1 of the 6 steps. Consequently, Colby does not teach "[a] method for upserting a source table with a destination table, the method comprising: selecting from a source table a first column comprising a plurality of elements; selecting from a destination table a second column comprising a plurality of elements; updating a row in the destination table with a row from the source table upon the success of a comparison operation on an element in the first column of the row from the source table and an element in the second column of the row from the destination table; and inserting a row from the source table into the destination table upon the failure of a comparison operation on an element in the first column of the row from the source table and an element in the second column of the row from the destination table, the method using no more than one query language statement" as recited in amended claim 9.

Cochrane and Colby, neither alone nor together, disclose, teach or suggest all the limitations in claim 9 and therefore, cannot be used to preclude patentability of claim 9 under U.S.C. § 103.

6. Claim 25 recited sufficiently similar limitations as claim 9 and as such is patentable over Cochrane and Colby for at least the same reason as claim 9.

7. Claims 10-12 and 26-28 depend on claims 9 and 25 and as such are patentable over Cochrane and Colby for at least the same reasons as those claims.

8. Claims 13, 21, and 29 share the common limitation with claim 9 of "using no more than one query language statement." As section A 5 of this paper shows that Cochrane and Colby do not disclose teach or suggest this limitation, Cochrane and Colby, neither alone nor together, disclose teach or suggest all the limitations in claims 13, 21, and 29. Therefore, Cochrane and Colby, neither alone nor together can be used to preclude patentability of claims 13, 21, or 29 under U.S.C. § 103.

9. Claims 14-16, 22-24, and 30-32 depend on claims 13, 21, and 29, and as such, are patentable over Cochrane and Colby for at least the same reasons as those claims.

III. Information Disclosure Statement

Applicants submitted an Information Disclosure Statement on January 18, 2005. However, we have not yet received confirmation that reference numbers 1-11 listed on form PTO/SB/08a and reference numbers 1-2 listed on form PTO/SB/08b have been initialed and considered. Attached hereto is a copy of the forms PTO/SB/08a (1 pg.) and PTO/SB/08b (1 pg.). Applicants hereby respectfully request that the references listed on forms PTO/SB/08a and PTO/SB/08b be initialed and considered by the Examiner.


CONCLUSION

On the basis of the above remarks, reconsideration and allowance of the claims is believed to be warranted and such action is respectfully requested. If the Examiner has any questions or comments, the Examiner is respectfully requested to contact the undersigned at the number listed below.

The Commissioner is authorized to charge any fees due in connection with the filing of this document to Bingham McCutchen's Deposit Account No. **50-2518**, referencing billing number **7011122001**. The Commissioner is authorized to credit any overpayment or to charge any underpayment to Bingham McCutchen's Deposit Account No. **50-2518**, referencing billing number **7011122001**.

Respectfully submitted,
Bingham McCutchen LLP

Dated: July 19, 2005

By: 
Gerald Chan
Reg. No. 51,541

Three Embarcadero Center, Suite 1800
San Francisco, CA 94111-4067
Telephone: (650) 849-4960
Telefax: (650) 849-4800